B.I.18

Algorithmen – Unterrichtseinheit

Java-Projekt: Flucht aus einem Irrgarten

Steffen Naundorf

Im Rahmen einer Projektarbeit lernen die Schülerinnen und Schüler Strategien zu entwerfen, wie sie selbst oder ein Computer aus einem Irrgarten entkommen können. Mithilfe einer Programmvorlage für *Java* werden sie angeleitet einen Algorithmus zu entwerfen, diesen zu testen und sukzessiv zu verbessern. Der Fokus liegt dabei auf dem Trainieren von prozessualem Denken.

KOMPETENZPROFIL – PROJEKTEINHEIT

Klassenstufe:	9/10 + Sek II
Dauer:	6–8 Unterrichtsstunden
Lernziele:	Die Lernenden … 1. implementieren einfache Befehle in *Java* zur Problemlösung „Entfliehen aus einem Irrgarten“, 2. implementieren einfache Schleifen, 3. analysieren und reflektieren den *Java* -Code mithilfe von systematischen Testungen und vorgegebenen Schemata, 4. vergleichen verschiedene Strategien.
Thematische Bereiche:	Algorithmen im Alltag, Programmierung, *Java, Entwicklungsumgebung, BlueJ*
Kompetenzbereiche:	Implementieren, Darstellen und Interpretieren, Probleme lösen und Handeln, Analysieren und Reflektieren

Auf einen Blick

Benötigte Materialien

- ☐ Computer mit Internetanschluss
- ☐ Entwicklungsumgebung für *Java* (z. B. *BlueJ*)
- ☐ *Java*-Projekt auf den Computern

Einstieg

Thema:	Mangel an Informationen als Herausforderung von Irrgärten
M 1	**Flucht aus einem Irrgarten – Blick aus verschiedenen Perspektiven**
Benötigt:	☐ Beamer/Whiteboard/Dokumentenkamera

Erarbeitung

Thema:	Aufsetzen des Projektes und Kennenlernen der Bedienung des Spiels
M 2	**Die Flucht aus dem Irrgarten**
Thema:	Wiederholung von Schleifen und Folgen eines geraden Gangs
M 3	**Hardcoden und Schleifen**
Thema:	Sackgassen und Kreuzungen
M 4	**Geschicktes Abbiegen**
Thema:	Die Linke-Hand-Regel
M 5	**Die Linke-Hand-Regel**
M 6	**Die Linke-Hand-Regel 2.0**
Benötigt:	https://www.youtube.com/watch?v=IIBwiGrUgzc
Thema:	Einfach und mehrfach verbundene Irrgärten
M 7	**Was ist ein Irrgarten und welche Arten gibt es**
Thema:	Vergleich von verschiedenen Strategien
M 8	**Strategien aus dem Irrgarten**
Benötigt:	Beispielimplementierungen

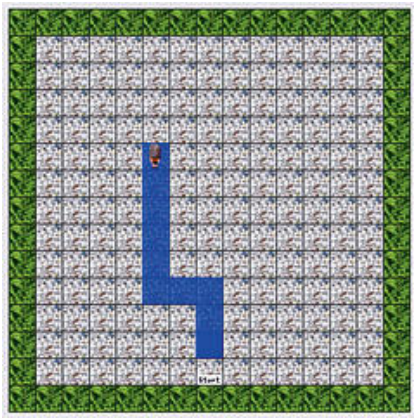
Benötigte Dateien

- ☐ *Canvas.java, Feld.java, Figur.java, Labyrinth.java, Spiel.java*
- ☐ *Linke-Hand-Regel.txt, Pledge-Algorithmus.txt*

Die Flucht aus dem Irrgarten

M 2

```
1  figur.gehe();
2  figur.gehe();
3  figur.gehe();
4  figur.dreheLinks();
5  figur.gehe();
6  figur.gehe();
7  figur.dreheRechts();
8  for (int i=0; i<5; i++) {
9      figur.gehe();
10 }
```



Navigation der Spielfigur mithilfe des nebenstehenden Codes

Steuern der Spielfigur

© pondomaniac/openga-meart.org/gemeinfrei

„Hallo, mein Name ist Lea und ich bin in diesem Irrgarten gefangen. Leider kenne ich den Weg nach draußen nicht. Hoffentlich könnt ihr mir helfen. Am Eingang zum Irrgarten befindet sich ein Computer, über den ihr mir sagen könnt, wie ich aus dem Labyrinth entkommen kann. Wenn ihr mir eine Anleitung schickt, was ich tun soll, kann ich so eventuell entkommen.
Sobald ich den Irrgarten betreten habe, könnt ihr mir auch mithilfe der Pfeiltasten Kommandos zurufen, wie ich laufen soll. Allerdings verschließt sich dann der Ausgang. Nur wenn ihr mir zu Anfang eure Anleitung gebt und ich nur mit deren Hilfe den Ausgang finde, komme ich heraus."

Nutzt für eure Anleitung die folgenden Kommandos. Ihr könnt ein Kommando auch an eine Bedingung knüpfen oder es in einer Schleife wiederholen. Eine ausführliche Beschreibung aller Funktionen (inkl. weiterer Klassen) findet ihr in der Projektdokumentation.

Kommando	Beschreibung
`figur.gehe();`	Ich gehe einen Schritt nach vorne.
`figur.dreheLinks();`	Ich drehe mich um 90° gegen den Uhrzeigersinn.
`figur.dreheRechts();`	Ich drehe mich um 90° im Uhrzeigersinn.
`figur.dreheNach(String richtung);`	Ich drehe mich in die entsprechende Richtung.
`figur.drehe180();`	Ich drehe mich einmal um.
`figur.istImZiel();`	Ich sage euch, ob ich im Ziel angekommen bin.
`figur.istVorneFrei();`	Ich sage euch, ob das Feld vor mir frei ist.
`figur.istHintenFrei();`	Ich sage euch, ob das Feld hinter mir frei ist.
`figur.istLinksFrei();`	Ich sage euch, ob das Feld links von mir frei ist.
`figur.istRechtsFrei();`	Ich sage euch, ob das Feld rechts von mir frei ist.
`figur.holeRichtung();`	Ich sage euch, in welche Richtung ich schaue.

M 4 Geschicktes Abbiegen

Bisher war der Weg für Lea recht einfach. Es war ein gerader Weg und das Ziel stets vor ihr. Nun wird es jedoch komplizierter und sie muss abbiegen. Die Frage ist nur wann?

Aufgaben

`level = „6“;`

1. Testet zunächst euren Code in diesem Level. Was fällt euch auf? Notiert Auffälligkeiten und Schwierigkeiten.
2. Überlegt und diskutiert, welche Informationen euch jeweils zur Verfügung. Notiert alle zur Verfügung stehenden Informationen, die euch an dieser Stelle helfen können.
   Tipp: Nehmt die Dokumentation zur Hilfe.
3. Passt eure Methode an, dass Lea das Ziel erreichen kann.
   Tipp: Solltet Ihr nicht weiterkommen, holt euch die Tippkarte M 4a.

`level = „7“;`

4. Testet euren Code mit diesem Level. Das Ziel ist verschoben, sodass ihr geradewegs in eine Sackgasse lauft. Überlegt, wie ihr Sackgassen erkennen könnt und darauf reagieren solltet. Passt euren Code entsprechend an.

`level = „8“;`

5. Nachdem ihr nun einem Weg folgen und auch aus einer Sackgasse entkommen könnt, geht es um das geschickte Abbiegen an Kreuzungen. Testet euren Algorithmus und schaut, was passiert. Diskutiert, was das Problem sein könnte.
   Überprüft eure Überlegungen mithilfe der Tippkarte **M 4d**.
6. Überlegt euch nun, wie Ihr mit verschiedenen Entscheidungsmöglichkeiten umgehen könnt.

M 6 Die Linke-Hand-Regel 2.0

In der Informatik gibt es häufig nicht die eine richtige Lösung für ein Problem. Stattdessen gibt es viele verschiedene Möglichkeiten ein Problem anzugehen bzw. einen Algorithmus zu entwickeln. Wie bei gesprochenen Sprachen ist es auch bei Programmiersprachen möglich, eine Botschaft unterschiedlich zu beschreiben.

```
4  while  (figur.istImZiel()==false)
   {
5   if (figur.istLinksFrei()) {
6     figur.dreheLinks();
7     figur.gehe();
8   } else if (figur.istVorneFrei())
    {
9     figur.gehe();
10  } else if (figur.istRechtsFrei())
   {
11    figur.dreheRechts();
12    figur.gehe();
13  } else {
14    figur.drehe180();
15    figur.gehe();
16  }
17 }
```

```
1  while  (figur.istImZiel()==false)
   {
2   if (figur.istLinksFrei()) {
3     figur.dreheLinks();
4     figur.gehe();
5   } else if (figur.istVorneFrei())
   {
6     figur.gehe();
7   } else {
8     figur.dreheRechts();
9   }
10 }
```


Aufgaben

`level = „9“;`

1. Die beiden Algorithmen oben zeigen eine Implementierung für die Linke-Hand-Regel. Beschreibt, worin sich die beiden Ansätze unterscheiden und begründet, warum der rechte Code ebenfalls funktioniert, obwohl er um einiges kürzer ist. Testet die beiden Ansätze.
   Überprüft eure Überlegungen mithilfe der Tippkarte **M 6a**.
2. Die beiden Algorithmen oben nehmen, wenn möglich immer den Weg links. Wäre ein spiegelbildlicher Algorithmus ebenfalls möglich? Hätte es Vor-/Nachteile?
   Überprüft eure Überlegungen mithilfe der Tippkarte 6b.